PyEMD Documentation

Release 0.2.11

Dawid Laszuk

Nov 25, 2020
# Table of Content

1 Intro  
1.1 General ................................................................. 3  
1.2 Installation .......................................................... 3  

2 Usage  
2.1 Typical Usage ....................................................... 5  
2.2 Parameters .......................................................... 5  

3 Speedup tricks  
3.1 Change data type .................................................... 7  
3.2 Change spline method ............................................. 8  
3.3 Decrease number of trials ....................................... 8  
3.4 Limit number of output IMFs .................................... 8  

4 Example  
4.1 EMD ................................................................. 9  
4.2 EEMD .............................................................. 11  

5 EMD ................................................................. 13  

6 EEMD  
6.1 Info ................................................................. 17  
6.2 Class .............................................................. 17  

7 BEMD  
7.1 Warning ............................................................ 21  
7.2 Info ................................................................. 21  
7.3 Class .............................................................. 21  

8 EMD2D  
8.1 Warning ............................................................ 23  
8.2 Info ................................................................. 23  
8.3 Class .............................................................. 23  

9 CEEMDAN  
9.1 Info ................................................................. 27  
9.2 Class .............................................................. 27
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Visualisation</td>
<td>31</td>
</tr>
<tr>
<td>11 Contact</td>
<td>33</td>
</tr>
<tr>
<td>12 Indices and tables</td>
<td>35</td>
</tr>
<tr>
<td>Bibliography</td>
<td>37</td>
</tr>
<tr>
<td>Index</td>
<td>39</td>
</tr>
</tbody>
</table>
Writing documentation is hard. If more clarifications are needed, or you think others might benefit from extra explanation, don’t hesitate to contact me through contact page.
1.1 General

*PyEMD* is a Python implementation of **Empirical Mode Decomposition (EMD)** and its variations. One of the most popular expansion is **Ensemble Empirical Mode Decomposition (EEMD)**, which utilises an ensemble of noise-assisted executions.

As a result of *EMD* one will obtain a set of components that possess oscillatory features. In case of plain *EMD* algorithm, these are called **Intrinsic Mode Functions (IMFs)** as they are expected to have a single mode. In contrary, *EEMD* will unlikely produce pure oscillations as the effects of injected noise can propagate throughout the decomposition.

1.2 Installation

1.2.1 Recommended

In order to get the newest version it is recommended to download source code from git repository. **Don’t worry, installation is simple**. Simply download this directory either directly from GitHub, or using command line:

```bash
$ git clone https://github.com/laszukdawid/PyEMD
```

Then go into the downloaded project and run from command line:

```bash
$ python setup.py install
```

1.2.2 PyPi (simplest)

Packaged obtained from PyPi is/will be slightly behind this project, so some features might not be the same. However, it seems to be the easiest/nicest way of installing any Python packages, so why not this one?
$ pip install EMD-signal
2.1 Typical Usage

Majority, if not all, methods follow the same usage pattern:

- Import method
- Initiate method
- Apply method on data

On vanilla EMD this is as

```python
from PyEMD import EMD
emd = EMD()
imfs = emd(s)
```

2.2 Parameters

The decomposition can be changed by adjusting parameters related to either sifting or stopping conditions.

2.2.1 Sifting

The sifting depends on the used method so these parameters ought to be looked within the methods. However, the typical parameters relate to spline method or the number of mirroring points.

2.2.2 Stopping conditions

All methods have the same two conditions, \textit{FIXE} and \textit{FIXE\_H}, for stopping which relate to the number of sifting iterations. Setting parameter \textit{FIXE} to any positive value will fix the number of iterations for each IMF to be exactly \textit{FIXE}. 
Example:

```python
emd = EMD()
emd.FIXE = 10
imfs = emd(s)
```

Parameter $FIXE_H$ relates to the number of iterations when the proto-IMF signal fulfils IMF conditions, i.e. number of extrema and zero-crossings differ at most by one and the mean is close to zero. This means that there will be at least $FIXE_H$ iteration per IMF.

Example:

```python
emd = EMD()
emd.FIXE_H = 5
imfs = emd(s)
```

When both $FIXE$ and $FIXE_H$ are 0 then other conditions are checked. These can be checking for convergence between consecutive iterations or whether the amplitude of output is below acceptable range.
CHAPTER 3

Speedup tricks

EMD is inherently slow with little chances on improving its performance. This is mainly due to it being a serial method. That’s both on within IMF stage, i.e. iterative sifting, or between IMFs, i.e. the next IMF depends on the previous. On top of that, the common configuration of the EMD uses the natural cubic spline to span envelopes, which in turn additionally decreases performance since it depends on all extrema in the signal.

Since the EMD is the basis for other methods like EEMD and CEEMDAN these will also suffer from the same problem. What’s more, these two methods perform the EMD many (hundreds) times which significantly increases any imperfections. It is expected that when it’ll take more than a minute to perform an EEMD/CEEMDAN with default settings on a 10k+ samples long signal with a "medium complexity". There are, however, a couple of tweaks one can do to do make the computation finish sooner.

Sections below describe a tweaks one can do to improve performance of the EMD. In short, these changes are:

- Change data type (downscale)
- Change spline method to piecewise
- Decrease number of trials
- Limit number of output IMFs

3.1 Change data type

Many programming frameworks by default casts numerical values to the largest data type it has. In case of Python’s Numpy that’s going to be numpy.float64. It’s unlikely that one needs such resolution when using EMD. A suggestion is to downcast your data, e.g. to float16. The PyEMD should handle the same data type without upcasting but it can be additionally enforce a specific data type. To enable data type enforcement one needs to pass the DTYPE, i.e.

---

0 I, the PyEMD’s author, will go even a bit further. If one needs such large resolution then the EMD is not suitable for them. The EMD is not robust. Hundreds of iterations make any small difference to be emphasised and potentially leading to a significant change in final decomposition. This is the reason for creating EEMD and CEEMDAN which add small perturbation in a hope that the ensemble provides a robust solution.
from PyEMD import EMD

emd = EMD(DTYPE=np.float16)

3.2 Change spline method

EMD was presented with the natural cubic spline method to span envelopes and that’s the default option in the PyEMD. It’s great for signals with not many extrema but its not suggested for longer/more complex signals. The suggestion is to change the spline method to some piecewise splines like ‘Akima’ or ‘piecewise cubic’.

Example:

```python
from PyEMD import EEMD
eemd = EEMD(spline_kind='akima')
```

3.3 Decrease number of trials

This relates more to EEMD and CEEMDAN since they perform an EMD a multiple times with slightly modified signal. It’s difficult to choose a correct number of iterations. This definitely relates to the signal in question. The more iterations the more certain that the solution is convergent but there is likely a point beyond which more evaluations change little. On the other side, the quicker we can get output the quicker we can use it.

In the PyEMD, the number of iterations is referred to by `trials` and it’s an explicit parameter to EEMD and CEEMDAN. The default value was selected arbitrarily and it’s most likely wrong. An example on updating it:

```python
from PyEMD import CEEMDAN
ceemdan = CEEMAN(trials=20)
```

3.4 Limit number of output IMFs

Each method, by default, will perform decomposition until all components are returned. However, many use cases only require the first component. One can limit the number of returned components by setting up an implicit variable `max_imf` to the desired value.

Example:

```python
from PyEMD import EEMD
eemd = EEMD(max_imfs=2)
```
Some examples can be found in PyEMD/example directory.

### 4.1 EMD

#### 4.1.1 Quick start

In most cases default settings are enough. Simply import `EMD` and pass your signal to `emd` method.

```python
from PyEMD import EMD
s = np.random.random(100)
emd = EMD()
IMFs = emd.emd(s)
```

**Something more**

Here is a complete script on how to create and plot results.

```python
from PyEMD import EMD
import numpy as np
import pylab as plt

# Define signal
s = np.linspace(0, 1, 200)
s = np.cos(11*2*np.pi*t*t) + 6*t*t

# Execute EMD on signal
IMF = EMD().emd(s,t)
N = IMF.shape[0]+1
```

(continues on next page)
The Figure below was produced with input:

\[ S(t) = \cos(22\pi t^2) + 6t^2 \]
4.2 EEMD

Simplest case of using ESNembld EMD (EEMD) is by importing `EEMD` and passing your signal to `eemd` method.

```python
from PyEMD import EEMD
import numpy as np
import pylab as plt

# Define signal
t = np.linspace(0, 1, 200)
sin = lambda x,p: np.sin(2*np.pi*x*t+p)
S = 3*sin(18,0.2)*(t-0.2)**2
S += 5*sin(11,2.7)
S += 3*sin(14,1.6)
S += 1*np.sin(4*2*np.pi*(t-0.8)**2)
S += t**2.1 - t

# Assign EEMD to `eemd` variable
eemd = EEMD()

# Say we want detect extrema using parabolic method
eemd.extrema_detection="parabol"

# Execute EEMD on S
eIMFs = eemd.eemd(S, t)
nIMFs = eIMFs.shape[0]

# Plot results
plt.figure(figsize=(12,9))
plt.subplot(nIMFs+1, 1, 1)
plt.plot(t, S, 'r')
for n in range(nIMFs):
    plt.subplot(nIMFs+1, 1, n+2)
    plt.plot(t, eIMFs[n], 'g')
    plt.ylabel("eIMF %i" % (n+1))
    plt.locator_params(axis='y', nbins=5)
plt.xlabel("Time [s]"
plt.tight_layout()
plt.savefig('eemd_example', dpi=120)
plt.show()
```
Empirical Mode Decomposition (EMD) is an iterative procedure which decomposes signal into a set of oscillatory components, called Intrinsic Mode Functions (IMFs).

```python
class PyEMD.EMD(spline_kind: str = 'cubic', nbsym: int = 2, **kwargs)
```

Empirical Mode Decomposition

Method of decomposing signal into Intrinsic Mode Functions (IMFs) based on algorithm presented in Huang et al. [Huang1998].

Algorithm was validated with Rilling et al. [Rilling2003] Matlab’s version from 3.2007.

Threshold which control the goodness of the decomposition:

- `std_thr` — Test for the proto-IMF how variance changes between siftings.
- `svar_thr` — Test for the proto-IMF how energy changes between siftings.
- `total_power_thr` — Test for the whole decomp how much of energy is solved.
- `range_thr` — Test for the whole decomp whether the difference is tiny.

References

[Huang1998], [Rilling2003]

Examples

```python
>>> import numpy as np
>>> T = np.linspace(0, 1, 100)
>>> S = np.sin(2*2*np.pi*T)
>>> emd = EMD(extrema_detection='parabol')
>>> IMFs = emd.emd(S)
>>> IMFs.shape
(1, 100)
```
__call__ (S: numpy.ndarray, T: Optional[numpy.ndarray] = None, max_imf: int = -1) →
numpy.ndarray
Call self as a function.

__init__ (spline_kind: str = 'cubic', nbsym: int = 2, **kwargs)
Initiate EMD instance.
Configuration, such as threshold values, can be passed as kwargs (keyword arguments).

Parameters
FIXE [int (default: 0)]
FIXE_H [int (default: 0)]
MAX_ITERATION [int (default 1000)] Maximum number of iterations per single sifting in EMD.
energy_ratio_thr [float (default: 0.2)] Threshold value on energy ratio per IMF check.
std_thr float [(default 0.2)] Threshold value on standard deviation per IMF check.
svar_thr float [(default 0.001)] Threshold value on scaled variance per IMF check.
total_power_thr [float (default 0.005)] Threshold value on total power per EMD decomposition.
range_thr [float (default 0.001)] Threshold for amplitude range (after scaling) per EMD decomposition.
extrema_detection [str (default ‘simple’)] Method used to finding extrema.
DTYPE [np.dtype (default np.float64)] Data type used.

>>> emd = EMD(std_thr=0.01, range_thr=0.05)

__weakref__ list of weak references to the object (if defined)

Huang criteria for IMF (similar to Cauchy convergence test). Signal is an IMF if consecutive siftings do not affect signal in a significant manner.

emd (S: numpy.ndarray, T: Optional[numpy.ndarray] = None, max_imf: int = -1) → numpy.ndarray
Performs Empirical Mode Decomposition on signal S. The decomposition is limited to max_imf imfs. Returns IMF functions in numpy array format.

Parameters
S [numpy array,] Input signal.
T [numpy array, (default: None)] Position or time array. If None passed or if self.extrema_detection == “simple”, then numpy arange is created.
max_imf [int, (default: -1)] IMF number to which decomposition should be performed. Negative value means all.

Returns
IMF [numpy array] Set of IMFs produced from input signal.

end_condition (S: numpy.ndarray, IMF: numpy.ndarray) → bool
Tests for end condition of whole EMD. The procedure will stop if:
• Absolute amplitude (max - min) is below range_thr threshold, or
• Metric L1 (mean absolute difference) is below \textit{total\_power\_thr} threshold.

**Parameters**

\textbf{S} [numpy array] Original signal on which EMD was performed.

\textbf{IMF} [numpy 2D array] Set of IMFs where each row is IMF. Their order is not important.

**Returns**

\textbf{end} [bool] Whether sifting is finished.

\textbf{extract\_max\_min\_spline} \((T: \text{numpy.ndarray}, \ S: \text{numpy.ndarray}) \rightarrow \text{Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]}\)

Extracts top and bottom envelopes based on the signal, which are constructed based on maxima and minima, respectively.

**Parameters**

\textbf{T} [numpy array] Position or time array.

\textbf{S} [numpy array] Input data \(S(T)\).

**Returns**

\textbf{max\_spline} [numpy array] Spline spanned on \(S\) maxima.

\textbf{min\_spline} [numpy array] Spline spanned on \(S\) minima.

\textbf{max\_extrema} [numpy array] Points indicating local maxima.

\textbf{min\_extrema} [numpy array] Points indicating local minima.

\textbf{find\_extrema} \((T: \text{numpy.ndarray}, \ S: \text{numpy.ndarray}) \rightarrow \text{Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]}\)

Returns extrema (minima and maxima) for given signal \(S\). Detection and definition of the extrema depends on \texttt{extrema\_detection} variable, set on initiation of EMD.

**Parameters**

\textbf{T} [numpy array] Position or time array.

\textbf{S} [numpy array] Input data \(S(T)\).

**Returns**

\textbf{local\_max\_pos} [numpy array] Position of local maxima.

\textbf{local\_max\_val} [numpy array] Values of local maxima.

\textbf{local\_min\_pos} [numpy array] Position of local minima.

\textbf{local\_min\_val} [numpy array] Values of local minima.

\textbf{get\_imfs\_and\_residue} () \rightarrow \text{Tuple[numpy.ndarray, numpy.ndarray]}\)

Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)

\textbf{prepare\_points} \((T: \text{numpy.ndarray}, \ S: \text{numpy.ndarray}, \ \text{max\_pos: numpy.ndarray, max\_val: numpy.ndarray, min\_pos: numpy.ndarray, min\_val: numpy.ndarray})\)

Performs extrapolation on edges by adding extra extrema, also known as mirroring signal. The number of added points depends on \texttt{nbym} variable.

**Parameters**

\textbf{S} [numpy array] Input signal.

\textbf{T} [numpy array] Position or time array.
max_pos  [iterable] Sorted time positions of maxima.
max_vali  [iterable] Signal values at max_pos positions.
min_pos  [iterable] Sorted time positions of minima.
min_val  [iterable] Signal values at min_pos positions.

Returns

min_extrema  [numpy array (2 rows)] Position (1st row) and values (2nd row) of minima.
min_extrema  [numpy array (2 rows)] Position (1st row) and values (2nd row) of maxima.
spline_points(T: numpy.ndarray, extrema: numpy.ndarray) → Tuple[numpy.ndarray, numpy.ndarray]
Constructs spline over given points.

Parameters

T  [numpy array] Position or time array.

extrema  [numpy array] Position (1st row) and values (2nd row) of points.

Returns

T  [numpy array] Position array (same as input).
spline  [numpy array] Spline array over given positions T.
6.1 Info

Ensemble empirical mode decomposition (EEMD) creates an ensemble of worker each of which performs an EMD on a copy of the input signal with added noise. When all workers finish their work a mean over all workers is considered as the true result.

6.2 Class

```python
class PyEMD.EEMD(trials: int = 100, noise_width: float = 0.05, ext_EMD=None, parallel: bool = False, **kwargs)
```

Ensemble Empirical Mode Decomposition

Ensemble empirical mode decomposition (EEMD) [Wu2009] is noise-assisted technique, which is meant to be more robust than simple Empirical Mode Decomposition (EMD). The robustness is checked by performing many decompositions on signals slightly perturbed from their initial position. In the grand average over all IMF results the noise will cancel each other out and the result is pure decomposition.

Parameters

- **trials** [int (default: 100)] Number of trials or EMD performance with added noise.
- **noise_width** [float (default: 0.05)] Standard deviation of Gaussian noise ($\hat{\sigma}$). It’s relative to absolute amplitude of the signal, i.e. $\hat{\sigma} = \sigma \cdot | \text{max}(S) - \text{min}(S) |$, where $\sigma$ is noise_width.
- **ext_EMD** [EMD (default: None)] One can pass EMD object defined outside, which will be used to compute IMF decompositions in each trial. If none is passed then EMD with default options is used.
- **parallel** [bool (default: False)] Flag whether to use multiprocessing in EEMD execution. Since each EMD(s+noise) is independent this should improve execution speed considerably. Note that it’s disabled by default because it’s the most common problem when EEMD takes too long time to finish. If you set the flag to True, make also sure to set `processes` to some reasonable value.
processes [int or None (optional)] Number of processes harness when executing in parallel mode. The value should be between 1 and max that depends on your hardware.

References

[Wu2009]

__call__ (S: numpy.ndarray, T: Optional[numpy.ndarray] = None, max_imf: int = -1) →
numpy.ndarray
Call self as a function.

__init__ (trials: int = 100, noise_width: float = 0.05, ext_EMD=None, parallel: bool = False, **kwargs)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

eemd (S: numpy.ndarray, T: Optional[numpy.ndarray] = None, max_imf: int = -1) →
numpy.ndarray
Performs EEMD on provided signal.

For a large number of iterations defined by trials attr the method performs emd () on a signal with added white noise.

Parameters

S [numpy array,] Input signal on which EEMD is performed.
T [numpy array or None, (default: None)] If none passed samples are numerated.
max_imf [int, (default: -1)] Defines up to how many IMFs each decomposition should be performed. By default (negative value) it decomposes all IMFs.

Returns
eIMF [numpy array] Set of ensemble IMFs produced from input signal. In general, these do not have to be, and most likely will not be, same as IMFs produced using EMD.

emd (S: numpy.ndarray, T: numpy.ndarray, max_imf: int = -1) →
numpy.ndarray
Vanilla EMD method.

Provides emd evaluation from provided EMD class. For reference please see PyEMD.EMD.

generate_noise (scale: float, size: Union[int, Sequence[int]]) →
numpy.ndarray
Generate noise with specified parameters. Currently supported distributions are:

- normal with std equal scale.
- uniform with range [-scale/2, scale/2].

Parameters

scale [float] Width for the distribution.
size [int] Number of generated samples.

Returns

noise [numpy array] Noise sampled from selected distribution.

get_imfs_and_residue () →
Tuple[numpy.ndarray, numpy.ndarray]
Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)
`noise_seed (seed: int) → None`
Set seed for noise generation.
7.1 Warning

Important This is an experimental module. Please use it with care as no guarantee can be given for obtaining reasonable results, or that they will be computed index the most computation optimal way.

7.2 Info

BEMD performed on bidimensional data such as images. This procedure uses morphological operators to detect regional maxima which are then used to span surface envelope with a radial basis function.

7.3 Class
8.1 Warning

Important This is an experimental module. Please use it with care as no guarantee can be given for obtaining reasonable results, or that they will be computed index the most computation optimal way.

8.2 Info

EMD performed on images. This version uses for envelopes 2D splines, which are span on extrema defined through maximum filter.

8.3 Class

class PyEMD.EMD2d.EMD2D(**config)

Empirical Mode Decomposition on images.

Important This is an experimental module. Experiments performed using this module didn’t provide acceptable results, either in actual output nor in computation performance. The author is not an expert in image processing so it’s very likely that the code could have been improved. Take your best shot.

Method decomposes images into 2D representations of loose Intrinsic Mode Functions (IMFs).

The current version of the algorithm detects local extrema, separately minima and maxima, and then connects them to create envelopes. These are then used to create a mean trend and subtracted from the input.

Threshold values that control goodness of the decomposition:

- `mseThr` — proto-IMF check whether small mean square error.
- `meanThr` — proto-IMF check whether small mean value.
__call__(image, max_imf=-1)
Call self as a function.

__init__(**config)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

check_proto_imf(proto_imf, proto_imf_prev, mean_env)
Check whether passed (proto) IMF is actual IMF. Current condition is solely based on checking whether
the mean is below threshold.

Parameters
* proto_imf [numpy 2D array] Current iteration of proto IMF.
* proto_imf_prev [numpy 2D array] Previous iteration of proto IMF.
* mean_env [numpy 2D array] Local mean computed from top and bottom envelopes.

Returns
* boolean Whether current proto IMF is actual IMF.

edm(image, max_imf=-1)
Performs EMD on input image with specified parameters.

Parameters
* image [numpy 2D array] Image which will be decomposed.
* max_imf [int, (default: -1)] IMF number to which decomposition should be performed.
  Negative value means all.

Returns
* IMFs [numpy 3D array] Set of IMFs in form of numpy array where the first dimension
  relates to IMF’s ordinary number.

classmethod end_condition(image, IMFs)
Determins whether decomposition should be stopped.

Parameters
* image [numpy 2D array] Input image which is decomposed.
* IMFs [numpy 3D array] Array for which first dimensions relates to respective IMF, i.e.
  (numIMFs, imageX, imageY).

evaluate_max_min_spline(image)
Calculates top and bottom envelopes for image.

Parameters
* image [numpy 2D array]

Returns
* min_env [numpy 2D array] Bottom envelope in form of an image.
* max_env [numpy 2D array] Top envelope in form of an image.

classmethod find_extrema(image)
Finds extrema, both minima and maxima, based on local maximum filter. Returns extrema in form of two
rows, where the first and second are positions of x and y, respectively.
Parameters

image [numpy 2D array] Monochromatic image or any 2D array.

Returns

min_peaks [numpy array] Minima positions.

max_peaks [numpy array] Maxima positions.

classmethod prepare_image(image)
Prepares image for edge extrapolation. Method bloats image by mirroring it along all axes. This turns extrapolation on edges into interpolation within bigger image.

Parameters

image [numpy 2D array] Image for which interpolation is required.

Returns

image [numpy 2D array] Big image based on the input. Grid 3x3 where the center block is input and neighbouring panels are respective mirror images.

classmethod spline_points(X, Y, Z, xi, yi)
Interpolates for given set of points
9.1 Info

Complete ensemble EMD with adaptive noise (CEEMDAN) performs an EEMD with the difference that the information about the noise is shared among all workers.

9.2 Class

```python
class PyEMD.CEEMDAN(trials: int = 100, epsilon: float = 0.005, ext_EMD=None, parallel: bool = False, **kwargs):
    "Complete Ensemble Empirical Mode Decomposition with Adaptive Noise"
    "Complete ensemble empirical mode decomposition with adaptive noise" (CEEMDAN) [Torres2011] is noise-assisted EMD technique. Word “complete” presumably refers to decomposing completely everything, even added perturbation (noise).

    Provided implementation contains proposed “improvements” from paper [Colominas2014].

    Any parameters can be updated directly on the instance or passed through a configuration dictionary.

    Goodness of the decomposition can be configured by modifying threshold values. Two are range_thr and total_power_thr which relate to the value range (max - min) and check for total power below, respectively.

    Parameters

    trials [int (default: 100)] Number of trials or EMD performance with added noise.
    epsilon [float (default: 0.005)] Scale for added noise (ε) which multiply std σ: β = ε · σ
    ext_EMD [EMD (default: None)] One can pass EMD object defined outside, which will be used to compute IMF decompositions in each trial. If none is passed then EMD with default options is used.
    parallel [bool (default: False)] Flag whether to use multiprocessing in EEMD execution. Since each EMD(s+noise) is independent this should improve execution speed considerably. Note
```
that it’s disabled by default because it’s the most common problem when CEEMDAN takes too long time to finish. If you set the flag to True, make also sure to set processes to some reasonable value.

**processes** [int or None (optional)] Number of processes harness when executing in parallel mode. The value should be between 1 and max that depends on your hardware.

**References**

[Torres2011], [Colominas2014]

```python
__call__(S: numpy.ndarray, T: Optional[numpy.ndarray] = None, max_imf: int = -1) -> numpy.ndarray
```

Call self as a function.

```python
__init__(trials: int = 100, epsilon: float = 0.005, ext_EMD=None, parallel: bool = False, **kwargs)
```

Configuration can be passed through config dictionary. For example, updating threshold would be through:

Example 1: >>> config = {“range_thr”: 0.001, “total_power_thr”: 0.01} >>> emd = EMD(**config)
Example 2: >>> emd = EMD(range_thr=0.001, total_power_thr=0.01)

```python
__weakref__
```

list of weak references to the object (if defined)

```python
emd(S: numpy.ndarray, T: numpy.ndarray, max_imf: int = -1) -> numpy.ndarray
```

Vanilla EMD method.

Provides emd evaluation from provided EMD class. For reference please see [PyEMD.EMD](#).

```python
def end_condition(S: numpy.ndarray, cIMFs: numpy.ndarray, max_imf: int) -> bool
```

Test for end condition of CEEMDAN.

Procedure stops if:

- number of components reach provided *max_imf*, or
- last component is close to being pure noise (range or power), or
- set of provided components reconstructs sufficiently input.

**Parameters**

- **S** [numpy array] Original signal on which CEEMDAN was performed.
- **cIMFs** [numpy 2D array] Set of cIMFs where each row is cIMF.
- **max_imf** [int] The maximum number of imfs to extract.

**Returns**

- **end** [bool] Whether to stop CEEMDAN.

```python
def generate_noise(scale: float, size: Union[int, Sequence[int]]) -> numpy.ndarray
```

Generate noise with specified parameters. Currently supported distributions are:

- **normal** with std equal scale.
- **uniform** with range [-scale/2, scale/2].

**Parameters**

- **scale** [float] Width for the distribution.
size [int or shape] Shape of the noise that is added. In case of int an array of that len is generated.

Returns

noise [numpy array] Noise sampled from selected distribution.

get_imfs_and_residue () ➔ Tuple[numpy.ndarray, numpy.ndarray]
  Provides access to separated imfs and residue from recently analysed signal. :return: (imfs, residue)

noise_seed (seed: int) ➔ None
  Set seed for noise generation.
A simple visualisation helper.

```python
class PyEMD.Visualisation (emd_instance=None)
    Simple visualisation helper.

    This class is for quick and simple result visualisation.

    plot_imfs (imfs=None, residue=None, t=None, include_residue=True)
        Plots and shows all IMFs.
        All parameters are optional since the emd object could have been passed when instantiating this object.
        The residual is an optional and can be excluded by setting include_residue=False.

    plot_instant_freq (t, imfs=None, order=False, alpha=None)
        Plots and shows instantaneous frequencies for all provided imfs.
        The necessary parameter is t which is the time array used to compute the EMD. One should pass imfs if no emd instances is passed when creating the Visualisation object.

    Parameters

        order [bool (default: False)] Represents whether the finite difference scheme is low-order
        (1st order forward scheme) or high-order (6th order compact scheme). The default value
        is False (low-order)

        alpha [float (default: None)] Filter intensity. Default value is None, which is equivalent to
        alpha = 0.5, meaning that no filter is applied. The alpha values must be in between -0.5
        (fully active) and 0.5 (no filter).
```
CHAPTER 11

Contact

Feel free to send email with any questions, concerns or for whatever reason you feel like.

**Email** \laszuk\dawid\@gmail.com (remove slashes \\).

**Homepage** http://www.laszukdawid.com

**GitHub** You can also visit PyEMD GitHub project page for this project.
CHAPTER 12

Indices and tables

- genindex
- search


Symbols

__call__() (PyEMD.CEEMDAN method), 28
__call__() (PyEMD.EEMD method), 18
__call__() (PyEMD.EMD method), 13
__call__() (PyEMD.EMD2d.EMD2D method), 23
__init__() (PyEMD.CEEMDAN method), 28
__init__() (PyEMD.EEMD method), 18
__init__() (PyEMD.EMD method), 14
__init__() (PyEMD.EMD2d.EMD2D method), 24
__weakref__ (PyEMD.CEEMDAN attribute), 28
__weakref__ (PyEMD.EEMD attribute), 18
__weakref__ (PyEMD.EMD attribute), 14
__weakref__ (PyEMD.EMD2d.EMD2D attribute), 24

C

CEEMDAN (class in PyEMD), 27
check_imf() (PyEMD.EMD method), 14
check_proto_imf() (PyEMD.EMD2d.EMD2D method), 24

E

EEMD (class in PyEMD), 17
eemd() (PyEMD.EEMD method), 18
EMD (class in PyEMD), 13
emd() (PyEMD.CEEMDAN method), 28
emd() (PyEMD.EEMD method), 18
emd() (PyEMD.EMD method), 14
emd() (PyEMD.EMD2d.EMD2D method), 24
EMD2D (class in PyEMD.EMD2d), 23
end_condition() (PyEMD.CEEMDAN method), 28
end_condition() (PyEMD.EMD method), 14
end_condition() (PyEMD.EMD2d.EMD2D method), 24
extract_max_min_spline() (PyEMD.EMD method), 15
extract_max_min_spline() (PyEMD.EMD2d.EMD2D method), 24

F

find_extrema() (PyEMD.EEMD method), 15
find_extrema() (PyEMD.EMD2d.EMD2D class method), 24

G

generate_noise() (PyEMD.CEEMDAN method), 28
generate_noise() (PyEMD.EEMD method), 18
get_imfs_and_residue() (PyEMD.CEEMDAN method), 29
get_imfs_and_residue() (PyEMD.EEMD method), 18
get_imfs_and_residue() (PyEMD.EMD method), 15

N

noise_seed() (PyEMD.CEEMDAN method), 29
noise_seed() (PyEMD.EEMD method), 18

P

plot_imfs() (PyEMD.Visualisation method), 31
plot_instant_freq() (PyEMD.Visualisation method), 31
prepare_image() (PyEMD.EMD2d.EMD2D class method), 25
prepare_points() (PyEMD.EMD method), 15

S

spline_points() (PyEMD.EMD method), 16
spline_points() (PyEMD.EMD2d.EMD2D class method), 25

V

Visualisation (class in PyEMD), 31